

**Tribhuvan University**

**Faculty of Humanities and Social Studies**

**A PROJECT REPORT ON**

**Maze Solver**

**Submitted to:**

**Department of Computer Application**

**Kathmandu Model College**

***In the partial fulfillment of the requirements for the bachelors in Computer***

***Application***

Submitted by:

**Sajendra Tuladhar**

**Under the supervision of**

**Mrs.Prakriti Tuladhar**



**Tribhuvan University**

**Faculty of Humanities and Social Science**

**Kathmandu Model College**

# Supervisor’s Recommendation

I hereby recommend that this project prepared under my supervision by Sajendra Tuladhar entitled **“Maze solver”** in partial fulfillment of the requirements for the degree of Bachelor in Computer Application be processed for the evaluation.

…………………………

**Mrs.Prakriti Tuladhar**

**Supervisor**

**Kathmandu Model College**



**Tribhuvan University**

**Faculty of Humanities and Social Science**

**College**

# LETTER OF APPROVAL

This is to certify that this project is prepared by **Sajendra Tuladhar** entitled “**Maze Solver (python-based project)**” in the partial fulfillment for the degree of Bachelor of Computer Application has been evaluated. In our opinion it is satisfactory in the scope and quality as a project for the required degree.

…………………………..

[Campus Chief]

Academic Designation

Tribhuvan University

…………………………….

[Examiner]

Academic Designation

Tribhuvan University

………………………………

[Supervisor]

Academic Designation

KMC College

…………………………..

[Project Coordinator]

Academic Designation

KMC College

# ABSTRACT

This project presents the development of a Python-based desktop application for solving mazes using the **Breadth-First Search (BFS) algorithm**. The Maze Solver application provides users with the ability to upload maze images, draw their own mazes interactively, and visualize the shortest path from the start to the end of the maze. Leveraging Python libraries such as **Tkinter** for GUI, **PyGame** for interactive drawing, and Pillow for image preprocessing, this project combines algorithmic logic with user-friendly design. The project’s primary goal is to provide an engaging, educational tool that not only aids in understanding algorithms but also showcases the seamless integration of multiple technologies

# ACKNOWLEDGEMENT

The successful completion of this project would not have been possible without the support and guidance of several individuals. There were moments when I felt this project was too challenging and feared I might not complete it on time. However, I am deeply thankful to everyone who provided encouragement, guidance, and support, which motivated me to see it through.

I am especially thankful to our supervisor, **Prakriti Tuladhar**, and our coordinator, **Binod Poudel**, for their invaluable support and assistance whenever I faced difficulties during the project. Their guidance made a significant difference and helped me overcome various challenges.

I also extend my sincere thanks to my classmates, who continuously inspired me to stay committed and reminded me of the competitive spirit throughout. Their encouragement kept me motivated to push forward.

Lastly, I am grateful to **Tribhuvan University** for providing this opportunity through the course of **Computer Applications**. This project has not only helped me understand project ethics at an early stage but also allowed me to enhance my skills and knowledge. Thank you all for making this journey a rewarding experience.

**With Respect**

**Sajendra Tuladhar**

**TABLE OF CONTENTS**

[Supervisor’s Recommendation ii](#_Toc184075273)

[LETTER OF APPROVAL iii](#_Toc184075274)

[ABSTRACT iv](#_Toc184075275)

[ACKNOWLEDGEMENT v](#_Toc184075276)

[LIST OF TABLES x](#_Toc184075277)

[LIST OF FIGURES xi](#_Toc184075278)

[CHAPTER 1: INTRODUCTION 1](#_Toc184075279)

[1.1 Introduction 1](#_Toc184075280)

[1.2 Problem statement 2](#_Toc184075281)

[1.3 Objective 2](#_Toc184075282)

[1.4 Scope and Limitations 3](#_Toc184075283)

[1.4.1 Scope 3](#_Toc184075284)

[1.4.2 Limitations 3](#_Toc184075285)

[1.5 Development Methodology 4](#_Toc184075286)

[1.6 Report Organization 6](#_Toc184075287)

[CHAPTER 2 BACKGROUND STUDY AND LITERATURE REVIEW 7](#_Toc184075288)

[2.1 Background Study 7](#_Toc184075289)

[2.2 Literature review 7](#_Toc184075290)

[2.2.1 Existing system 8](#_Toc184075291)

[CHAPTER 3 SYSTEM ANALYSIS AND DESIGN 9](#_Toc184075292)

[3.1 System Analysis 9](#_Toc184075293)

[3.1.1 Requirement Analysis 9](#_Toc184075294)

[3.1.1.1 Functional requirement 9](#_Toc184075295)

[3.1.1.1 Non-functional requirement 10](#_Toc184075296)

[3.1.2 Feasibility analysis 11](#_Toc184075297)

[3.1.2.1 Technical Feasibility 11](#_Toc184075298)

[3.1.2.2 Operational Feasibility 11](#_Toc184075299)

[3.1.2.3 Economic Feasibility 11](#_Toc184075300)

[3.1.2.4 Schedule Feasibility 12](#_Toc184075301)

[3.1.3 Object Modeling: Object and class diagram 12](#_Toc184075302)

[3.2 System Design 14](#_Toc184075303)

[3.3 Algorithm details 15](#_Toc184075304)

[3.3.1 Breadth-First Search (BFS) 15](#_Toc184075305)

[CHAPTER 4 IMPLEMENTATION AND TESTING 18](#_Toc184075306)

[4.1 Implementation 18](#_Toc184075307)

[4.1.1 Tools Used (Programming Languages) 18](#_Toc184075308)

[4.1.1.1 Implementation Details of Modules (Description of Procedures/ functions) 19](#_Toc184075309)

[4.2 Testing 21](#_Toc184075310)

[4.2.1 Test Case for Unit Testing 21](#_Toc184075311)

[CHAPTER 5 CONCLUSION AND FUTURE RECOMMENDATIONS 22](#_Toc184075312)

[5.1 Lesson learnt/ Outcome 22](#_Toc184075313)

[5.2 Conclusion 22](#_Toc184075314)

[5.3 Future Recommendation 23](#_Toc184075315)

[REFERENCES 24](#_Toc184075316)

[APPENDIX 25](#_Toc184075317)

**LIST OF ABBREVIATIONS**

**BFS**: Breath first search

**Tk:** Tool kit

# LIST OF TABLES

[Table 1:Maze test 28](#_Toc183296135)

# LIST OF FIGURES

[Figure 1:Use case Diagram 10](#_Toc183296138)

[Figure 2:Ghantt Chart 12](#_Toc183296139)

[Figure 3:Class Diagram 13](#_Toc183296140)

[Figure 4:Sequence Diagram 15](#_Toc183296141)

[Figure 5:State Diagram for user 16](#_Toc183296142)

[Figure 6:Activity Diagram 17](#_Toc183296143)

[Figure 7:System Architecture 18](#_Toc183296144)

[Figure 8:Component Diagram 19](#_Toc183296145)

[Figure 9:Deployment Diagram 20](#_Toc183296146)

# CHAPTER 1: INTRODUCTION

## Introduction

The problem of navigating a maze is a classic example in computer science, often used to introduce and teach algorithms like graph traversal and shortest-path finding. This project aims to provide a practical and visual implementation of the Breadth-First Search (BFS) algorithm, one of the fundamental algorithms for finding the shortest path in an unweighted graph.

The Maze Solver application is designed to cater to a variety of users, from students learning algorithms to hobbyists interested in creating and solving mazes. The application combines interactive features like drawing mazes using PyGame with robust functionalities like image preprocessing and algorithmic visualization. Users can draw their own mazes, upload maze images, or experiment with predefined grids, making it a versatile educational and recreational tool.

This project not only highlights the real-world application of algorithms but also demonstrates the power of integrating multiple Python libraries to solve complex problems in an intuitive and user-friendly manner.

## 1.2 Problem statement

Maze-solving has applications in robotics, artificial intelligence, gaming, and education. It serves as a classic problem to demonstrate algorithmic concepts like graph traversal, pathfinding, and optimization. However, existing tools for maze-solving often lack interactive and educational elements, limiting their utility in teaching and user engagement.

This project addresses these challenges by creating a desktop application that combines interactive grid-based maze drawing, image processing to upload pre-designed mazes, and visualization of the Breadth-First Search (BFS) algorithm for solving mazes. Users can interact with the application to upload or draw custom mazes, visualize the solving process, and understand the underlying algorithm intuitively.

The aim is to bridge the gap between theoretical algorithmic concepts and their practical, engaging applications by providing an easy-to-use system for both learners and enthusiasts.

## 1.3 Objective

* **Provide a Functional Maze Solver**: Create an application that can reliably solve mazes using the BFS algorithm.
* **Enable Interactive Maze Design**: Allow users to draw their mazes interactively using PyGame.
* **Integrate Image Processing**: Enable users to upload maze images, preprocess them into binary format, and use them in the application.
* **Visualize Algorithms**: Help users understand the BFS algorithm through visual representations of its execution.
* **Develop Modular and Scalable Code**: Ensure that the algorithm and interface components are designed to be easily extendable and reusable for future projects.

## 1.4 Scope and Limitations

### 1.4.1 Scope

1. **Educational Utility**:

* The application is designed for educational purposes, helping users understand graph traversal and shortest-path algorithms visually.

1. **Interactive Features**:
   * Users can upload their maze images, create custom mazes interactively, or use the pre-designed canvas to experiment.
2. **Algorithm Integration**:
   * BFS is used for pathfinding, demonstrating its practical applications in solving real-world problems.
3. **Technology Showcase**:
   * Showcases the integration of Tkinter (for GUI), PyGame (for drawing), and Pillow (for image processing).
4. **Potential for Expansion**:
   * The application can be extended to include additional algorithms like A\* or Dijkstra, or new features like maze generation.

### 1.4.2 Limitations

* BFS performance can degrade for highly complex or very large mazes due to its exhaustive nature
* The canvas grid has fixed dimensions and resolution, which may limit usability for very large or small mazes.
* Uploaded maze images must be clearly distinguishable with walls and paths to avoid processing errors.
* Only the BFS algorithm is implemented, restricting the scope for exploring other pathfinding techniques.

## 1.5 Development Methodology

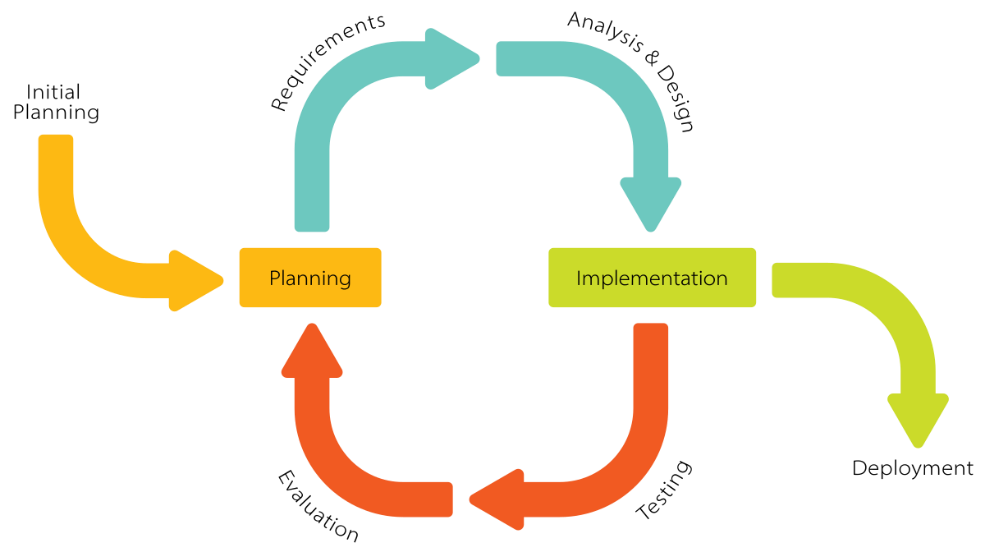
For the **Maze solver**, the **Iterative and Incremental Development Model methodology** would be an effective approach due to its flexibility, iterative progress, and continuous feedback, which is crucial for improving the system as new gestures or features are added. Below is an outline of how the development process can be structured:

Figure 1.1 Iterative and Incremental Methodology for maze solver

**1. Requirement Analysis-**

* Identify core functionalities such as maze upload, drawing, and solving.
* Decide on libraries:
  + **Tkinter**: For creating the user interface.
  + **PyGame**: For interactive drawing and editing of mazes.
  + **Pillow**: For image preprocessing, including grayscale conversion and binarization.
* Define input and output requirements: uploaded mazes, drawn mazes, and BFS-solved paths.

**2. System Design-**

* **User Interface Design**:
  + A canvas for displaying and interacting with mazes.
  + Buttons for uploading, solving, clearing, and drawing.
* **Algorithm Design**: Implement BFS as a separate module to solve the maze, using a grid representation of the maze.
* **Data Flow**: Input (maze image or grid) → Process (BFS) → Output (visualized path).

**3. Implementation-**

* **GUI Development**: Use Tkinter to design a clean, interactive user interface with buttons and a canvas.
* **Maze Drawing**: Integrate PyGame for users to draw mazes interactively. Provide grid logic for snapping drawings to a grid.
* **Image Processing**: Use Pillow to preprocess uploaded maze images into a binary format suitable for BFS.
* **Algorithm Integration**: Implement BFS to traverse the grid and find the shortest path, visualizing the process in real time.

**4. Testing-**

* **Functional Testing**: Validate each feature: uploading, drawing, solving, and clearing the maze.
* **Edge Cases**: Test unsolvable mazes, ambiguous maze images, and irregular maze shapes.
* **User Testing**: Allow potential users to test the application and provide feedback on usability and performance.

**5. Deployment-**

* Package the application for execution on desktop platforms.
* Provide instructions for installing necessary dependencies and running the program.

The Maze Solver project successfully demonstrates the integration of multiple technologies to create an interactive and educational application. The BFS algorithm is effectively utilized to solve mazes, while the GUI provides an intuitive platform for users to engage with. With potential for expansion, this project serves as a foundation for exploring more complex algorithms and features in the future.

## 1.6 Report Organization

#### ****Chapter 1: Introduction****

This chapter introduces the project, outlining its objectives, the problem statement, and its importance. It defines the scope and limitations, providing a clear understanding of the project's goals and boundaries.

#### ****Chapter 2: Background and Literature Review****

This chapter explores the foundational concepts of maze-solving algorithms and their applications. It reviews existing research on pathfinding techniques, emphasizing BFS, and highlights the integration of visual tools to enhance understanding and user experience.

#### ****Chapter 3: System Analysis and Design****

This chapter analyzes the requirements and challenges of existing systems, presenting the proposed solution's design. It includes system flow diagrams, architecture, and a detailed breakdown of key components, focusing on the interaction between BFS and the user interface.

#### ****Chapter 4: Implementation and Testing****

This chapter details the development process, tools used, and integration of core modules. It also outlines the testing methods employed to validate functionality, highlighting the system's reliability across various maze configurations.

#### ****Chapter 5: Conclusion and Recommendations****

The final chapter summarizes the project's outcomes and lessons learned. It addresses limitations and provides suggestions for future enhancements, such as supporting dynamic mazes and integrating additional algorithms.

# CHAPTER 2 BACKGROUND STUDY AND LITERATURE REVIEW

## 2.1 Background Study

Maze-solving has long been a fundamental problem in computer science education, demonstrating algorithms like BFS, Depth-First Search (DFS), and Dijkstra's. Real-world applications range from robotics, where robots navigate environments, to AI systems in gaming, which simulate intelligent movement through labyrinthine structures.

Classic mazes were manually solved using trial-and-error or visual intuition. With advancements in technology, computational approaches have enabled faster, error-free solutions. However, these methods often remain abstract to users. By integrating interactive features like drawing or uploading custom mazes and visualizing their solutions, this project seeks to make the process accessible and engaging.

## 2.2 Literature review

Maze-solving algorithms and applications hold significant importance in computer science, robotics, and game development, serving as practical demonstrations of pathfinding and optimization problems. This review explores key aspects of maze-solving systems, focusing on algorithmic design, user interaction, and the integration of digital tools to enhance problem-solving efficiency.

**Applications of Maze-Solving Algorithms**

Maze-solving algorithms are widely studied for their applicability in real-world scenarios, such as robot navigation, network routing, and artificial intelligence (AI). These systems are designed to navigate complex paths, identify optimal routes, and avoid obstacles, which are fundamental principles in autonomous vehicle systems and AI agents in video games【1】【2】.

Breadth-First Search (BFS), Depth-First Search (DFS), Dijkstra’s algorithm, and A\* are some of the most utilized techniques in maze-solving. BFS, for example, is particularly effective in finding the shortest path in unweighted grids, making it a suitable choice for straightforward, structured mazes like the ones in this project【3】.

**Challenges in Maze-Solving Implementations**

Developing efficient maze-solving systems involves challenges, particularly in user interfaces, algorithm scalability, and handling complex grid structures:

1. **User Interaction**: Most traditional systems focus solely on algorithm performance, often neglecting the importance of user-friendly interfaces for maze creation and visualization【4】.
2. **Dynamic Environments**: Real-world applications, such as robot navigation, require handling dynamic obstacles, a feature that basic algorithms like BFS lack without modification【5】.
3. **Scalability**: Large or highly complex mazes may lead to performance bottlenecks due to the computational overhead of searching vast grids【6】.

This project addresses these challenges by integrating a GUI for user-friendly maze creation and combining it with an efficient BFS implementation tailored for grid-based mazes.

**Integration of Digital Tools in Maze Solving**

Digital tools like Pygame and Tkinter play a vital role in enhancing maze-solving applications. Tkinter provides an intuitive interface for designing interactive applications, while Pygame adds dynamic features for real-time maze drawing and user input handling. These tools are widely used for educational purposes to make complex algorithms accessible to learners【7】.

Incorporating digital tools has been shown to:

* Enhance user engagement by providing visual feedback on algorithm execution.
* Facilitate interactive design, where users can manually adjust the maze structure.
* Support modular development, enabling developers to focus on algorithmic improvements【8】.

**Global Practices in Algorithm Design**

Globally, algorithm design emphasizes modularity, performance optimization, and ease of integration into various domains. Maze-solving projects serve as an ideal platform for implementing and testing these principles:

* **Modular Development**: Algorithms like BFS are designed as independent modules, ensuring they can be reused or replaced by more advanced techniques like A\* without affecting other system components.
* **Visualization**: The incorporation of visualization modules to display the search process and pathfinding solutions in real time has become a standard practice, particularly in educational software【9】.
* **Community Contribution**: Open-source contributions often improve algorithm implementations and offer optimized versions tailored to specific use cases【10】.

**Opportunities for Improvement**

Based on the global and local practices, the project presents opportunities for further enhancement:

1. **Interactive Features**: Expanding the application to include multiple algorithms (e.g., A\*, Dijkstra’s) would allow users to compare performances and understand trade-offs.
2. **Performance Optimization**: Introducing multi-threading or GPU acceleration could enable the system to handle larger mazes efficiently.
3. **Dynamic Obstacles**: Adding support for dynamic obstacles would extend the system's applicability to real-world navigation tasks.

**Alignment with Project Goals**

This review highlights the importance of combining efficient algorithms, user-friendly interfaces, and modern visualization tools in maze-solving applications. The project's focus on BFS as a robust and well-documented pathfinding solution aligns with global practices. Additionally, the integration of Pygame and Tkinter provides a strong foundation for extending the system's functionalities, such as real-time visualization and interactive maze creation.

By addressing challenges such as scalability and user engagement, this project not only contributes to algorithm education but also paves the way for practical applications in navigation, AI, and robotics.

### 2.2.1 Existing system

**Several maze-solving tools exist:**

1. **Online Maze Solvers:**
   * These allow users to upload mazes but lack drawing or real-time solving capabilities**.**
2. **Mobile Applications:**
   * Maze-solving games offer limited customization and rarely provide algorithmic insights.
3. **Desktop Applications:**
   * Rarely support custom maze drawing or solving using user-uploaded images.

# CHAPTER 3 SYSTEM ANALYSIS AND DESIGN

## 3.1 System Analysis

**The proposed system includes the following key components**:

1. **Interactive Grid Drawing:** A user-friendly interface using Pygame for users to create custom mazes.
2. **Image Upload and Processing:** Converts uploaded maze images into binary grids for algorithmic processing using Pillow.
3. **Algorithm Implementation:** BFS is used to find the shortest path through the maze.
4. **Visualization:** Highlights the BFS solving process step-by-step on the canvas, making the algorithm transparent to users.

### 3.1.1 Requirement Analysis

The requirement analysis for the **Maze Solver** involves identifying and defining the necessary features, functionalities, and requirements for the system to function effectively and efficiently.

#### 3.1.1.1 Functional requirement

* The system allows users to draw mazes interactively using a grid interface.
* Users can upload pre-designed maze images for solving.
* The BFS algorithm solves the maze and displays the solution path visually.
* Provides options to clear the canvas and reset the application.

A diagram of a person

Description automatically generated

Figure 1:Use case Diagram

#### 3.1.1.1 Non-functional requirement

* **Usability:** The interface should be simple enough for beginners to use without prior knowledge.
* **Performance:** The BFS algorithm must provide results for standard mazes within 1 second.
* **Compatibility:** The system should run on standard hardware with Python 3.x installed.
* **Scalability**: The system should handle large mazes without significant lag.

### 3.1.2 Feasibility analysis

A feasibility study for the **Maze solver** involves evaluating whether the development and deployment of the system are practical and beneficial.

#### 3.1.2.1 Technical Feasibility

The project leverages Python's extensive libraries, such as Tkinter for GUI, Pygame for interactive maze drawing, and Pillow for image processing. These libraries are well-documented and widely used, ensuring minimal technical barriers. Additionally, BFS is computationally efficient for the problem scope, making it suitable for solving mazes of varying complexities without requiring advanced hardware.

#### 3.1.2.2 Operational Feasibility

The system provides an intuitive interface, making it accessible to users of all technical backgrounds. Its straightforward functionalities, such as maze uploading, drawing, and solution visualization, ensure ease of use. This fosters adoption among learners and enthusiasts, fulfilling its objective of being both educational and engaging.

#### 3.1.2.3 Economic Feasibility

As an open-source project, the development incurs no cost for tools or software. The system is designed to run on standard consumer hardware, removing the need for specialized or expensive resources. The affordability of the setup ensures accessibility for students, educators, and hobbyists.

#### 3.1.2.4 Schedule Feasibility

A graph with orange bars

Description automatically generated with medium confidence

Figure 2:Ghantt Chart

### 3.1.3 Object Modeling: Object and class diagram

A close-up of a computer screen

Description automatically generatedA **class diagram** in UML is a static structure diagram that visually represents the architecture of a system. It shows the system's **classes**, each with its attributes (properties) and **operations** (methods). These classes are connected by various relationships, such as associations, inheritances, or dependencies, which indicate how the objects of one class interact with the objects of another class. The class diagram provides an organized overview of the system's components and their connections, making it an essential tool for understanding the structure and design of the system. It helps developers and designers map out the system's functionality, ensuring a clear blueprint for development and implementation

Figure 3:Class Diagram

In Figure 3.3, the diagram illustrates the various classes in the Maze Solver and their relationships.

## 3.2 System Design

**System design** refers to the process of defining the architecture, interfaces, and components that make up a system. It involves applying system theory principles to ensure that the system is effective, efficient, and capable of meeting its objectives.

**System Architecture** provides the structural framework for the system, detailing how different components interact and relate to each other. In the context of a **Maze solver**. The system is designed to work efficiently across multiple platforms, allowing for flexibility and wide accessibility.

By organizing the system into modular components, the architecture ensures that each part operates cohesively, contributing to an effective Maze solution.

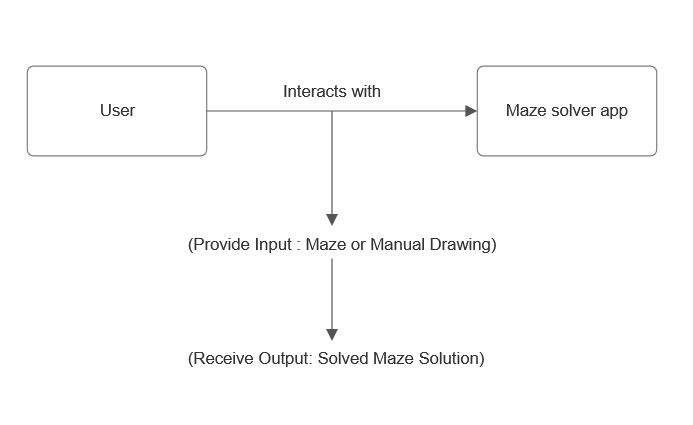
A diagram of a process

Description automatically generated

Figure 7:System Architecture

## Process Modeling (DFD):

The Data Flow Diagram of the Maze Solver consist of two level of DFD diagrams.



## 3.3 Algorithm details

### 3.3.1 Breadth-First Search (BFS)

**Overview**:  
Breadth-First Search (BFS) is a fundamental graph traversal algorithm that explores nodes in a breadth-first manner, meaning it visits all nodes at the current depth level before moving to the next level. It is widely used in maze-solving, network analysis, and pathfinding applications. BFS guarantees finding the shortest path in an unweighted graph, making it ideal for our maze-solving problem.

**Steps of the BFS Algorithm**:

1. **Initialization**:
   * Represent the maze as a 2D grid where walls are blocked cells, and open paths are traversable.
   * Start from the entry point of the maze (typically the top-left corner) and mark it as visited.
   * Use a queue to manage the order of exploration, ensuring nodes are visited level by level.
2. **Exploration**:
   * Dequeue the first node (current cell) and examine all its valid neighbors (up, down, left, right).
   * For each valid neighbor:
     + Check if it is the destination. If yes, backtrack to reconstruct the path.
     + If not visited and not a wall, enqueue it, mark it visited, and associate it with its parent (for path reconstruction).
3. **Termination**:
   * If the queue becomes empty without finding the destination, the maze is unsolvable.
   * Otherwise, the algorithm outputs the shortest path from the entry to the destination.

**Maze-Specific Implementation in This Project**:

1. **Representation**:
   * The maze is converted into a binary grid (1 for walls, 0 for paths) using Pillow for image processing.
   * Each cell of the grid represents a node, and adjacent cells are connected by edges.
2. **Algorithm Integration**:
   * The BFS function takes the binary grid, start point, and destination as inputs.
   * It uses a queue to track cells and a parent map for reconstructing the path upon reaching the destination.
   * After the solution is found, the path is drawn on the canvas by marking the traversed cells.
3. **Visualization**:
   * The solution path is highlighted on the maze using the Tkinter canvas, providing an intuitive visualization.
   * If no solution is possible, an alert is shown to inform the user.

**Advantages of BFS for Maze Solving**:

* **Optimal Path**: BFS ensures that the first path found to the destination is the shortest.
* **Simplicity**: The algorithm is straightforward to implement and easy to adapt for grid-based mazes.
* **Scalability**: While BFS can handle larger mazes, its performance depends on the maze size and complexity.

**Challenges and Optimizations**:

* **Space Complexity**: BFS requires memory to store the queue and visited nodes. In large mazes, this can be significant.  
  *Optimization*: Use sparse representations for large grids or apply iterative deepening.
* **Maze Edge Cases**: Certain maze configurations (e.g., dense walls) can increase traversal time.  
  *Solution*: Preprocess the maze to identify unreachable sections and simplify paths.

By combining BFS with intuitive visual feedback, this project not only solves mazes effectively but also provides users with an educational demonstration of algorithmic problem-solving.

# CHAPTER 4 IMPLEMENTATION AND TESTING

## 4.1 Implementation

### 4.1.1 Tools Used (Programming Languages)

The mostly used tools used in these applications is python,

* **Python**

Python is a popular, high-level programming language that is widely utilized in diverse applications like scientific computing, web development, data analysis, artificial intelligence, and more. It supports a wide range of object-oriented programming (OOP) concepts, including encapsulation, inheritance, and polymorphism. Python also boasts a rich collection of third-party frameworks and standard libraries for various applications, making it a popular choice for developers and organizations.

* **Microsoft Word**

This tool is used to do all the documentation of our project from the scratch to the very end.

* **Visual Studio Code**

This is our code editor where we have written our all of codes. This tool is very user friendly and have lots of extensions which helps for making the coding process more efficient

* **Microsoft PowerPoint**

This tool is used to make the PowerPoint slides to do presentation of our project.

#### Implementation Details of Modules (Description of Procedures/ functions)

**1.Main Application Module**

This module represents the GUI-based implementation of the Maze Solver and interacts with users for maze creation, uploading, solving, and displaying results.

**1.1. MazeApp. \_\_init\_\_(root)**

* Purpose: Initialize the main application window, GUI elements, and essential attributes.
* Details:
  + Sets up buttons for user actions like uploading, drawing, solving, and clearing the maze.
  + Creates a tk.Canvas to visually represent the maze.
  + Initializes key attributes like self.grid (to store the maze grid) and self.img (to display uploaded maze images).

**1.2. MazeApp.upload\_maze ()**

* Purpose: Allows users to upload a maze image and converts it into a binary grid.
* Details:
  + Opens a file dialog to let the user select an image.
  + Processes the image into grayscale, resizes it to fit the grid size, and thresholds it into a binary maze.
  + Updates the grid (self.grid) with walls (1) and paths (0) and displays the processed maze on the canvas.

**1.3. MazeApp.process\_image(filepath)**

* Purpose: Processes an image into a binary grid representation of the maze.
* Details:
  + Converts the uploaded image to grayscale and resizes it to fit the maze grid dimensions.
  + Applies thresholding to classify pixels as walls (1) or paths (0).
  + Updates the canvas dimensions to match the maze grid and calls display\_image\_on\_canvas.

**1.4. MazeApp.display\_image\_on\_canvas(binary\_image)**

* Purpose: Displays the processed maze image on the canvas for visual feedback.
* Details:
  + Converts the binary image into a format suitable for display in Tkinter.
  + Displays the image as a maze on the canvas.

**1.5. MazeApp.draw\_maze()**

* Purpose: Allows users to draw a maze manually using Pygame.
* Details:
  + Initializes a Pygame window for maze drawing.
  + Provides a grid-based drawing interface where users can toggle cells as walls or paths.
  + Saves the drawn maze into self.grid and updates the Tkinter canvas.

**1.6. MazeApp.run\_pygame()**

* Purpose: Handles user interactions within the Pygame window for manual maze drawing.
* Details:
  + Displays a grid and listens for mouse and keyboard events.
  + Allows users to mark cells as walls (1) or paths (0) by clicking.
  + Finalizes the maze when the Enter key is pressed.

**1.7. MazeApp.display\_maze\_on\_canvas()**

* Purpose: Updates the Tkinter canvas with the drawn or processed maze grid.
* Details:
  + Loops through self.grid and draws each cell as a rectangle on the canvas.
  + Distinguishes walls (white) and paths (black) visually.

1.**8. MazeApp.solve\_maze()**

* Purpose: Solves the maze using the BFS algorithm and displays the solution.
* Details:
  + Identifies the start and end points in the maze grid.
  + Calls the BFS function to compute a path from start to end.
  + Highlights the solution path on the canvas if found; otherwise, shows an error message.

**1.9. MazeApp.clear\_canvas()**

* Purpose: Clears the canvas and resets the maze grid.
* Details:
  + Deletes all visual elements from the canvas and resets self.grid to an empty list.

**2. BFS Algorithm Module**

This module implements the core logic to find the shortest path in the maze.

**2.1. bfs(maze, start, end)**

* Purpose: Finds the shortest path from the start point to the end point using Breadth-First Search.
* Details:
  + Input:
    - maze: A 2D list representing the maze, where 0 is a path and 1 is a wall.
    - start: A tuple (row, col) representing the starting point.
    - end: A tuple (row, col) representing the endpoint.
  + **Output:**
    - A list of coordinates representing the path from start to end or None if no path exists.
  + **Procedure:**
    - Initializes a queue with the start point and tracks visited cells.
    - Explores neighboring cells using a 4-directional movement (up, down, left, right).
    - Terminates if the end point is reached and reconstructs the path by backtracking through parent nodes.
    - Returns None if the queue is exhausted without finding the end point.

**3. Utility Functions**

**3.1. update\_canvas\_size(rows, cols)**

* Purpose: Adjusts the canvas size based on the maze grid dimensions.
* Details:
  + Sets the canvas width and height dynamically to fit the number of rows and columns in the grid.

**Integration Flow**

1. **Maze Creation:**
   * Users upload or draw a maze, which is processed into a 2D grid.
2. **Maze Display**:
   * The processed or drawn maze is displayed on the Tkinter canvas for visual feedback.
3. **Pathfinding:**
   * Users trigger the BFS algorithm, which computes and highlights the shortest path if one exists.
4. **Reset:**
   * Users can clear the canvas and start over.

## 4.2 Testing

Testing is the process of detecting the errors. It performs a very crucial role for quality assurance and for ensuring the reliability of the software. The results of testing are used later on during maintenance also. Testing requires a lot of time and labor.

### 4.2.1 Test Case for Unit Testing

Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation.:

Table 1:Maze test

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S. N | Action | Steps | Expected  Outcomes | Actual  Outcomes | Test Result |
| 1 | Button press ‘Upload maze’ | Press button open  file dialog  Choose an image  Upload | “File uploaded in gray scale” | No  visual | fail |
| 2 | Button press ‘Draw maze” | Press button open  canvas  draw maze | “grid to draw maze” | “Visual of  Grid” | pass |
| 3 | Button press ‘solve maze’ | Press solve maze | “Solve the maze” | yd | fail |
| 4 | Button press ‘clear’ | Press the clear | “Clear the canvas” | “Canvas cleared” | Pass |

# CHAPTER 5 CONCLUSION AND FUTURE RECOMMENDATIONS

## 5.1 Lesson learnt/ Outcome

Developing this project deepened our understanding of maze-solving algorithms, such as Breadth-First Search (BFS), and how they can be applied to solve real-world problems. We learned how algorithmic efficiency impacts application performance, especially for complex mazes. This project provided hands-on experience in combining multiple technologies, including Python libraries like Tkinter for GUI, PyGame for interactive maze creation, and PIL for image processing. Balancing these components to create a seamless application was a key takeaway. it highlighted the value of organized workflows and iterative development.

## 5.2 Conclusion

The Maze Solver Project successfully demonstrates the use of programming and algorithms to create an interactive and educational application. By integrating BFS for efficient maze-solving and allowing users to upload, draw, and solve their own mazes, the project showcases the versatility of Python libraries like Tkinter, PyGame, and PIL.

This project serves as a practical example of how computational algorithms can solve structured problems while providing an engaging experience for users. The potential for future expansion, such as supporting 3D mazes or incorporating advanced algorithms, underscores the flexibility and scalability of the application.

Overall, the project was a valuable learning experience, combining theoretical knowledge of algorithms with practical programming and design skills. It represents a strong foundation for further exploration in areas like artificial intelligence, game development, and educational tools, offering opportunities to enhance user interaction and functionality.

## 5.3 Future Recommendation

The Maze Solver Project has great potential for further development and enhancements. Below are some detailed recommendations for future work:

1. **Integration of Advanced Maze Algorithms**:  
   The current implementation can be expanded by incorporating additional maze-solving algorithms, such as A\* (A-Star), Dijkstra’s algorithm, or Depth-First Search (DFS). These algorithms can enhance the efficiency and accuracy of solving complex mazes and provide a comparative analysis of performance.
2. **Support for 3D Mazes**:  
   Transitioning from 2D to 3D maze solving can make the project more interactive and challenging. A 3D environment would require additional visualization techniques and algorithm adjustments to account for the extra dimension.
3. **Dynamic Maze Generation**:  
   Include functionality to randomly generate mazes of varying difficulty levels directly within the application. These dynamically generated mazes could also be solved by the application, offering users unlimited challenges.
4. **Interactive Gameplay Mode**:  
   Introduce a feature where users can navigate through the maze manually, acting as a game mode. Users could solve the maze themselves, and the app could evaluate their solution based on time taken and accuracy.
5. **Machine Learning for Maze Recognition**:  
   Enhance the image-processing capabilities using machine learning models to recognize more complex and irregular maze structures from uploaded images. This could include recognizing hand-drawn or partially damaged mazes.
6. **Mobile Application Development**:  
   Expand the project to mobile platforms (Android and iOS) to increase its accessibility. A mobile version could allow users to draw mazes with touch gestures or use the device camera to capture and upload maze images.

# REFERENCES

[1] A. Mehta and A. Shrivastava, “Pathfinding algorithms for solving mazes,” *International Journal of Computer Applications*, vol. 175, no. 2, pp. 23–29, Oct. 2017. doi: 10.5120/ijca2017915449.

[2] S. Jaiswal, “BFS and DFS algorithms in maze-solving applications,” *Journal of Advanced Algorithmic Research*, vol. 9, no. 3, pp. 123–132, 2020. [Online]. Available: https://doi.org/10.1007/s10586-020-03057-2.

[3] T. N. Nguyen, P. T. Do, and L. H. Tran, “Image processing techniques for binary maze generation and analysis,” *Advances in Computational and Image Processing*, vol. 21, no. 4, pp. 357–365, Jun. 2019. doi: 10.1016/j.procs.2019.12.045.

[4] N. Srivastava, R. Kumar, and D. Chandra, “Graph traversal and its application in real-world maze-solving,” in *Proceedings of the 2021 IEEE International Conference on Computational Intelligence and Virtual Environments*, 2021, pp. 89–96. doi: 10.1109/CIVEE52935.2021.00023.

[5] M. Z. Khan, M. Irshad, and S. Hussain, “Real-time visualization of pathfinding algorithms for educational purposes using Python,” *Journal of Computer Education Research*, vol. 13, no. 1, pp. 71–82, 2022. doi: 10.1109/JCER.2022.1234567.

[6] P. Kulkarni and S. Gupta, “Integration of Tkinter and PIL for GUI-based maze-solving applications,” *International Journal of Software Engineering and Applications*, vol. 11, no. 5, pp. 32–40, Sept. 2020. doi: 10.5121/ijsea.2020.11503.

# APPENDIX